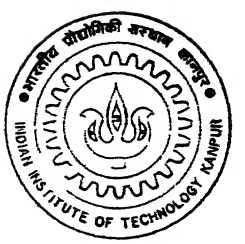
DATA COMPRESSION USING BOOLEAN FUNCTION FORMULATION

by

P. Balakrishna Reddy



DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

APRIL, 1996

DATA COMPRESSION USING BOOLEAN FUNCTION FORMULATION

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

P. Balakrishna Reddy

to the

DEPARTMENT OF
ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

April 1996

CERTIFICATE

2/4/96

It is certified that the thesis entitled *'DATA COMPRESSION USING BOOLEAN FUNCTION FORMULATION"* by **P. BALAKRISHNA REDDY** has been carried out under my supervision and that it has not been submitted elsewhere for the award of a degree

(**Dr. M. U. Siddiqi**)

Professor

Dept of Electrical Engg

IIT Kanpur, INDIA – 208016

April 1996

Dedicated to

My Mother

Acknowledgements

I express my deep sense of gratitude and appreciation to my thesis supervisor **Dr. M. U. Siddiqi** for his guidance and encouragement I thank him for giving the problem, which is in my area of interest

It is time to recall the moments spent with *Ravindra* and *Basha* during my 20 months stay

My special thanks to the authorities of IIT Kanpur who make this institution an excellent learning place

P. B. K. Reddy

Abstract

A novel approach for lossless as well as lossy compression of monochrome images using partitioning of Boolean functions by Walsh-Hadamard Transform and by minimizing Boolean functions in Reed Muller representation form is presented The image is split into bit planes, and the bit-planes are divided into blocks Each block is transformed into a Boolean switching function, treating the pixel value as output of the function In Walsh-Hadamard case compression is performed by mapping the Boolean function to a prototype Boolean function and coding the operations required for it using adaptive block coding method by treating the operations as blocks In Reed-Muller case, compression is achieved by minimizing the switching function by selecting the appropriate basis and by coding the coefficients and the basis Our technique of lossless compression involves linear prediction or indexing method as a preprocessing step, and has compression ratio greater than block coding lossless compression technique Lossless compression technique has been extended to text files also Our lossy compression technique involves reducing the number of bit planes as a preprocessing step, which incurs minimal loss in the information of the image The bit planes that remain after preprocessing are compressed by using lossless compression technique Qualitatively one cannot distinguish between the original image and the lossy image and the value of the mean square error is kept low The compression scheme is slower while the decompression is comparable to that of block coding

Contents

List of Figures

List of Tables

Definitions

1 A Boolean switching function F is a mapping $F \quad B^{\setminus} \rightarrow B$ where B = {0 1}

2 In the truth table of a switching function of N variables there are $2^{\setminus}$ rows Each row represents a input vector called min term

3 For a switching function the ON-set is the set of Reed-Muller coefficients having value 1 and the OFF-set having value 0

4 For Walsh-Hadamard transform, we define switching function F is a mapping $F \quad B^{\setminus} \rightarrow B$ where B = {1, −1}

5 The performance of a lossless data compression algorithm is asserted by the parameters compression ratio and compression time We define compression ratio as

$$\frac{\text{total input bytes - total output bytes}}{\text{total input bytes}} \times 100\%$$

6 bit-plane Consider an image, where each pixel is represented by k-bits By selecting a single bit from the same position in the binary representation of each pixel, k-binary images called bit planes can be generated

7 Uncomplimented literal is represented as x_i

Complimented literal is represented as $\overline{x}_i$

x_i' represents either complimented or uncomplimented literal

Chapter 1

Introduction

Compression of Image data is important from the point of view of storage and transmission, as it helps in saving storage space and transmission time Lossless compression techniques permit the recovery of an exact copy of original image where as lossy techniques permit only an approximation to be recovered Conventional lossless compression techniques employ de-correlation techniques such as DPCM or Hierarchical Interpolation followed by a coding scheme such as Huffman coding or arithmetic coding In this thesis we propose two different approaches to code sub blocks of an image bit-plane

1 By converting each Boolean function into its prototype Boolean function by Walsh Hadamard Transform and coding the operations required for it

2 Minimizing the Boolean function by Reed-Muller representation and coding the coefficients

In lossless compression the image file after being preprocessed is split into bit-planes Each bit plane is a binary image which is divided into fixed blocks, and each block is converted into a Boolean function The functions are then converted into its prototype Boolean function by Walsh-Hadamard Transform The operations required for it are coded to obtain the compressed data In Reed-Muller Representation, Boolean functions which are converted from blocks are minimized by representing them in the appropriate basis functions The minimized function which represents coefficients having either only ones or only zeros are coded to obtain the compression

In lossy compression we reduce the number of bit levels in the original image to less than or equal to 32 by applying center of mass technique The image containing at most 32 gray levels will be recoded by mapping the gray values to a 5 bit gray code, thus reducing the number of bit planes from 8 to 5 The mapping function leads to a fixed over head of only 96 bits These five bit planes will be processed in the same manner as we do for the lossless compression scheme

1.1 Compression schemes

In this section we are discussing the different compression schemes

- In Block coding, image is divided into bit planes and bit planes are divided into blocks, the all 0(white) and all 1(black) blocks are represented by 2-bit code, where as mixed type block is stored directly after a 2-bit code [6] Results of this block coding on images is shown in results chapter Compression of images can be done with the combination of block coding and arithmetic coding [7] with compressed block are represented by a two bit header and the coded bits, Uncompressed block is represented by a two bit header and the original bits

- JPEG employees a de-correlation scheme based on DPCM(several predictors are provided) followed by adaptive Huffman coding or arithmetic coding The lossy JPEG processes are based on the Discrete Cosine Transform(DCT) and entropy coding of the quantized DCT coefficients based on adaptive Huffman or arithmetic coding The DCT turns an array of intensity data into an array of frequency data that tells how fast the intensity vary Quantization sets the precision to which each of the values resulting from the DCT is stored JPEG uses linear quantization which means, that each of the DCT values are divided by a quantization factor and rounded to an integer to get the value that will be stored These quantized values are coded by adaptive Huffman or arithmetic coding Decompression reverses the above steps

- In run length coding [13], a series of repeated values(pixel values) is replaced by a single value and a count. For example a series of values like abbbbbbbc-cddddeeddd can be replaced by coding the count of consecutive letters Images

with large areas of constant shade, this scheme gives higher compression Run length coding is used in many bit map file formats(TIFF)

- Huffman coding is a common scheme for compression It works by substituting more efficient codes for data In this scheme binary code is assigned to to each unique value, with the codes varying in length These assignments are stored in a conversion table which is sent to the decoding software before the codes are sent For file compression compression ratios are order of 8 1 can be achieved Huffman coding is poor choice for files with long runs of single values which can be better compressed using run-length or other codings Huffman coding also needs accurate statistics, on how frequently each value occurs in the original file Huffman coding is done in two passes In the first pass, the statistical model is created, in the second the data are encoded As a result, and because variable-length codes require a lot of processing to decode, Huffman compression and decompression is a relatively slow process

- In arithmetic compression, uses shorter codes for frequently occurring things and longer codes for infrequently occurring things [11] Arithmetic compression comes close to theoretical limits for compression This involves mapping every different sequence to a region on an imaginary line between 0 and 1 That region is represented as a binary fraction of variable precision(number of bits) Therefore less common data require a higher precision(more bits) Arithmetic compression can reduce file size dramatically, depending on the source and the accuracy of the statistical model used Compressions of 100 1 may be achieved

- Lossless data compression is performed in computers by using the following methods [4] Some of the commands which are used in the computers, and their internal operation is described below The command compress, which is used in our computer systems uses an adaptive Lempel-Ziv coding [8] This coding will be done by replacing 9 bit codes (257 and up) to common substrings in the file When code 512 is reached, the algorithm switches to 10 bit codes and continuos to use more bits until the limit specified by the user or by default 16 After max-bits limits is reached, compress periodically checks the compression ratio If

it is increasing, compress continuos to use the existing code dictionary If it is decreasing compress discards the table of strings and rebuilds it from the scratch this allows the algorithm to adapt to the next block of file

The command pack uses Huffman coding on a byte by byte basis The amount of compression obtained depends upon the size of the input file and the character frequency distribution

1.2 Organization of thesis

In chapter 2 we describe the methods which have been adopted to solve our problem In chapter 3 we describe the implementation of our lossless lossy algorithms In chapter 4 we have given the results obtained for our algorithm In chapter 5 we give the conclusions and the future scope of work

Chapter 2

Mathematical Preliminaries

In this chapter we discuss the two methods which have been adapted to solve our thesis problem *Walsh-Hadamard* transform has been used to classify Boolean functions, using five invariance operations We derive the effect of those five invariance operations on spectrum *Reed-Muller* expansion has been used to minimize the Boolean function with appropriate polarity function

2.1 Walsh Hadamard transform

Basis functions for Walsh Hadamard transform [3] are
$W_u(x) = (-1)^{(\sum_{i=0}^{n-1} u_i x_i)}$
where w_i and x_i are binary representations of x, w
w,x $\in$ 0 to $2^n - 1$ w and x being,

$$w = w_{n-1}2^{n-1} + w_{n-2}2^{n-2} + \quad + w_0$$

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \quad + x_0$$

Walsh-Hadamard transform of a n variable Boolean function f(x) is given by

- $s_{f(x)}(w) = \sum_{x=0}^{2^n-1} f(x)(-1)^{\sum_{i=0}^{n-1} u_i x_i}$
- $s_{f(x)}(w) = \sum_{x=0}^{2^n-1} f(x)(-1)^{<w \quad x>}$
- $\underline{S} = \underline{\underline{W}} \quad \underline{F}$

where, $\underline{\underline{W}}$ is $2^n \times 2^n$ matrix

$\underline{F}$ is $2^n \times 1$ matrix whose elements are real values 1 and -1 with binary 0 mapped to 1 and binary 1 to -1

and $\underline{S}$ is $2^n \times 1$ spectral coefficient matrix $< u\, x >$ is called inner product which is equal to $\underline{w}^T \ \underline{x}$ where, $\underline{w}\ \underline{x}$ are coloumn vectors Each valid spectrum represents a Boolean function, and requires $2^n \times 2^{n-1}$ number of operations(additions or subtractions) to compute it for a n variable Boolean function Walsh-Hadamard matrices are orthogonal symmetric, and its inverse is identical to itself with multiplying factor $\frac{1}{2^n}$ So that

$$\underline{\underline{W}}^{-1} = \frac{1}{2^n}\underline{\underline{W}}$$

Walsh-Hadamard matrix for n = 3, is shown below

$$\underline{\underline{W}} = \left[\begin{array}{cccccccc|l} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & x_0 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & x_1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & x_0 \oplus x_1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & x_2 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & x_0 \oplus x_2 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & x_1 \oplus x_2 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & x_0 \oplus x_1 \oplus x_2 \end{array}\right]$$

By reading 1 as 0 and -1 as 1 in the hadamard matrix, row vectors may be seen to related to x_i input variables over $(0\ \ 2^n - 1)$ as shown in the matrix above The resulting spectrum coefficients in $\underline{S}$ is equal to the number of occasions the min term value of the Boolean function agree with the corresponding Hadamard row values minus the number of times the two values disagree Representing $\underline{S} =< S_0, S_1, S_2, \quad , S_{2^n-1} >$ by $< S_0, S_1, S_2, S_{12}, S_3, S_{13}, \quad , S_{123\ \ n} >$ where S_{ij} represents the correlation between the Boolean function and $x_i \oplus x_j$

S_0 being the zeroth order coefficient

S_i i = 1 to n the primary or first order coefficient,

S_{ij} ij = 12, 13 $\cdots$ the second order coefficient,

S_{ijk} ijk = 123, 124, 134 the third order coefficient, and so on

The Walsh-Hadamard matrix $\underline{\underline{W}}$ of the order $2^n \times 2^n$ it can be decomposed and is

equal to $\underline{\underline{M}}^n$ For n = 3,

$$\underline{\underline{M}} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

We used this matrix to find the spectrum of a binary function, by this the number of operations(additions or subtractions) will be reduced to $n < 2^n$

The properties of the spectral coefficients obtained for the Boolean functions

1 The sum of all spectral coefficients of S of a fully defined Boolean function is equal to 2^n.

2 The maximum value of a spectral coefficient of S is $\pm 2^n$, this occurs when f(x) is equal to any row of the Hadamard matrix or negative of it The range of each coefficient is $-2^n, -2^n + 2, \quad , 0, \quad , 2^n - 2, 2^n$.

3 When any of the spectrum coefficient is maximum valued, then all other coefficients are zero

4 When any input variable x_i is redundant in the function, then the spectral coefficients having subscript i, will be zero valued

2.2 Classification of Boolean functions

The set of 2^{2^n} Boolean functions of less than or equal to n variables can be classified into equivalent classes as shown in figure 2 1 In each equivalent class all the functions can be realised using any Boolean function in that class [1][2] Classification of Boolean functions has two advantages

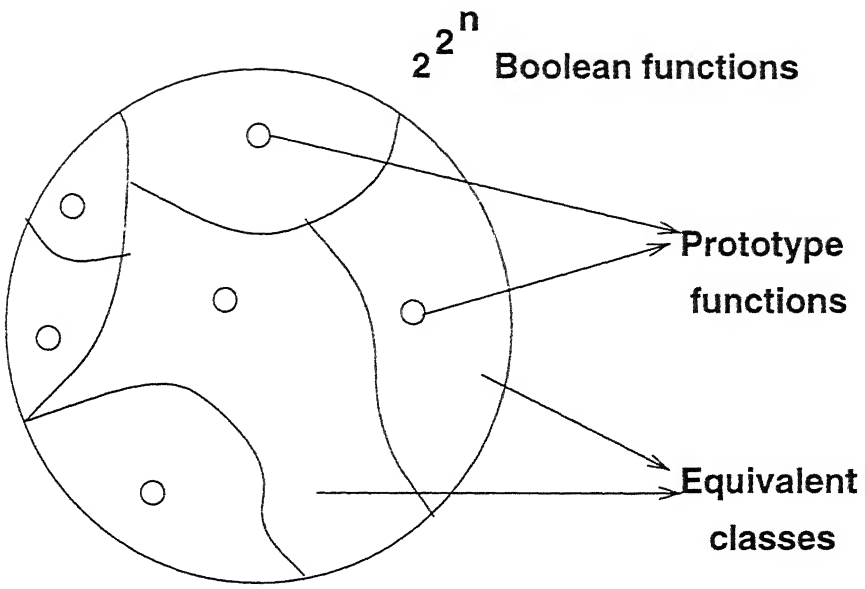


Figure 2 1 Classification of Boolean functions

1 Testing and fault diagnosis procedures may be standardised for each classification entry as they have identical realization

2 One prototype or standard Boolean function from each classification entry is used to realize all other functions, from appropriate operations corresponding to the classification procedure

In classifying the Boolean functions we do the following five basic operations

1 (N) Negation of one or more input variables x_i

2 (P) Permutation of two or more input variables x_i

3 (N) Negation of the output function $\overline{f(x)}$

4 Replacing one or more input variables by Exclusive-or of that variable and one or more other input variables

5 Replacing function output by Exclusive-or of that function and one or more input variables

Boolean functions which are classified under the first three operations are called NPN equivalent functions
for example

$$f_1(X) = x_1\overline{x}_2 + x_2x_3$$
$$f_2(X) = x_1x_2 + \overline{x}_2x_3$$
$$f_3(X) = x_1x_3 + x_2\overline{x}_3$$
$$f_4(X) = \overline{x}_1x_3 + \overline{x}_2\overline{x}_3$$

functions $f_1(X)$ and $f_2(X)$ are related by the input permutation $x_1 \leftrightarrow x_2$, functions $f_1(X)$ and $f_2(X)$ are related by the negation $x_2 \rightarrow \overline{x}_2$ and the permutation $x_2 \leftrightarrow x_3$ and functions $f_3(X)$ and $f_4(X)$ are negation of each other $f_3(X) = \overline{f}_4(X)$, resulting these four functions fall into the same NPN classification

One more algebraic classification called Self dual algebraic classification is much more compact than NPN equivalent classification The dual of any function f(X) is given by $f^D(X) = \overline{f(\overline{X})}$, where bar in the parentheses indicates that all the input variables x_i are complemented If the dual of any function is equal to the function it self then that function is called self dual function The property of a self dual function $f^{SD}(X_n)$ of n variable is, when it is decompossed into two functions about any variable x_i, the two resulting $n-1$ variable functions will be duals of each other The function $f^{SD}(X_{n+1})$ being a representative function for all the NPN equivalent functions obtained by decomposing $f^{SD}(X_{n+1})$ about each of its x_i input variables, i = 1 to n+1 Thus $f^{SD}(X_{n+1})$ is much more compact than NPN classification

2.3 Spectral classification

In this section we discuss the five basic operations in classifying Boolean functions of less than or equal to n variables in the transform domain.

2.3.1 Effect of invariance operations on spectrum of Boolean functions

We derive the effect of five invariance operations, which are mentioned above, to full set of 2^n spectral coefficients

1 *Interchange(permutation) of input variables* x_i *and* x_j

Let f(x) be a Boolean function and S(w) be the spectrum

where x = $(x_{n-1}, x_{n-2}, \quad , x_0)$

and w = $(w_{n-1} \ w_{n-2}, \quad . w_0)$

From the definition a Walsh-Hadamard transform of a Boolean function

$S(w) = \sum_{x=0}^{2^n-1} f(x)(-1)^{\sum_{i=0}^{n-1} w_i x_i}$

$S(w) = \sum_{x=0}^{2^n-1} f(x)(-1)^{<w x>}$

Let S'(w) be the spectrum of a Boolean function $f(\underline{\underline{\sigma}} \ \underline{x})$

where $\underline{\underline{\sigma}}$ is a $n \times n$ non singular matrix with elements from Z_2 and the additive operation is with respect to mod 2 From the definition of WHT

$$S'(w) = \sum_{x=0}^{2^n-1} f(\underline{\underline{\sigma}})(-1)^{<\underline{w} \ \underline{x}>} \tag{2 1}$$

where $< \underline{w} \ \underline{x} >$ is called inner product and is equal to $\underline{w}^T \underline{x}$

Let $\underline{y} = \underline{\underline{\sigma}} \ \underline{x}$, substituting this in the equation (2 1) we have

$$S'(w) = \sum_{x=0}^{2^n-1} f(\underline{y})(-1)^{<\underline{w} \ \underline{\underline{\sigma}}^{-1} \underline{y}>} \tag{2 2}$$

consider the term $< \underline{w}, \underline{\underline{\sigma}}^{-1} \underline{y} >$, which is equal to

$$\underline{w}^T (\underline{\underline{\sigma}}^{-1} \underline{y})$$

it can be written as

$$\underline{u}^T \underline{\underline{\sigma}}^{-1} \underline{y}$$

further it can be written as

$$(\underline{\underline{\sigma}}^{-1^T} \underline{w})^T \underline{y} \tag{2 3}$$

(Here, we made use of matrix properties, $(AB)^T = B^T A^T$ and $A^{T^T} = A$ where A and B are matrices)

substituting equation (2 3) in equation (2 2) we have

$$\sum_{x=0}^{2^n-1} f(x)(-1)^{<\underline{\underline{\sigma}}^{-1^T} \underline{w}, \underline{x}>} \tag{2 4}$$

From the definition of WHT

$$S'(u) = S'\underline{\underline{\sigma}}^{-1^T}\underline{u} \tag{2.5}$$

For example the matrix shown below corresponds to a four variable Boolean function $f(x_3, x_2, x_1, x_0)$

$$\underline{\underline{\sigma}} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

The corresponding new input variables are x_0 is replaced by $x_0 \oplus x_1$ x_1 and x_2 are interchanged(permuted), x_3 is replaced by $x_2 \oplus x_3$

By interchanging the input variables x_i and x_j, $i \neq j$ $i, j \in 1$ to n, the resulting new spectrum from the original Boolean function spectrum can be obtained by replacing the subscript i by subscript j and vice versa This requires the interchange of 2^{n-2} pairs of spectral coefficients

$$s_i \leftrightarrow s_j$$

$$s_{ik} \leftrightarrow s_{jk}$$

$$s_{ikl} \leftrightarrow s_{jkl}$$

Note that the coefficients s_0, s_k, s_{ij} · remain unchanged

2 *Complementation of input variable* x_i

Let S(w) be the spectrum of f($x_{n-1}, x_{n-2}, \quad , x_i, \quad , x_0$)

and S'(w) be the spectrum of f($x_{n-1}, x_{n-2}, \quad , \overline{x}_i, \quad , x_0$)

Now, from the definition of WHT,

$$S(w) = \sum_{x=0}^{2^n-1} f(x)(-1)^{\sum_{j=0}^{n-1} w_j x_j} \tag{2.6}$$

and S / (w) is given by

$$S'(w) = \sum_{x=0}^{2^n-1} f(x_{n-1}, x_{n-2}, \quad , \overline{x}_i, \quad , x_0)(-1)^{<wx>} \tag{2.7}$$

Partitioning the space 0 to $2^n - 1$ into two parts p and $\overline{p}$ where

$p \in 0$ to $2^n - 1$ for all values of x_i is equal to zero

$\overline{p} \in 0$ to $2^n - 1$ for all values of x_i is equal to one

therefore equation (2 6) can be written as

$$S(w) = \sum_{x \in p} f(x)(-1)^{\sum_{j=0}^{n-1} w_j x_j} + \sum_{x \in \overline{p}} f(x)(-1)^{\sum_{j=0}^{n-1} w_j x_j} \qquad (2\ 8)$$

in equation (2 8) the term $(-1)^{\sum_{j=0}^{n-1} w_j x_j}$ can be written as $(-1)^{\sum_{j \neq i} w_j x_j}(-1)^{w_i x_i}$ substituting this in equation (2 8) we have,

$$S(w) = \sum_{x \in p} f(x)(-1)^{\sum_{j \neq i} w_j x_j}(-1)^{w_i x_i} + \sum_{x \in \overline{p}} f(x)(-1)^{\sum_{j \neq i} w_j x_j}(-1)^{w_i x_i} \qquad (2\ 9)$$

In the equation (2 9), x_i is equal to 0 in the first term and x_i is equal to 1 in the second term Therefore we have,

$$S(w) = \sum_{x \in p} f(x)(-1)^{\sum_{j \neq i} w_j x_j} + (-1)^{w_i} \sum_{x \in \overline{p}} f(x)(-1)^{\sum_{j \neq i} w_j x_j} \qquad (2\ 10)$$

Let y = 0001000 $\mp x$ where 1 is in i'th position,

substituting this in equation(2 7) we have,

$$S'(w) = \sum_{y=0}^{2^n-1} f(y)(-1)^{\sum_{j \neq i} w_j x_j}(-1)^{w_i \overline{y}_i} \qquad (2\ 11)$$

Again dividing the space 0 to $2^n - 1$ and following the same steps from equation (2 6) to (2 10) we have,

$$S'(w) = \sum_{y \in p} f(y)(-1)^{\sum_{j \neq i} w_j y_j}(-1)^{w_i \overline{y}_i} + \sum_{y \in \overline{p}} f(y)(-1)^{\sum_{j \neq i} w_j x_j}(-1)^{w_i \overline{y}_i} \qquad (2\ 12)$$

In the first term in equation (2 12) $\overline{y}_i$ is equal to 1 and for the second term $\overline{y}_i$ is equal to 0 Substituting these values in equation(2 12) we have,

$$S'(w) = (-1)^{w_i} \sum_{y \in p} f(y)(-1)^{\sum_{j \neq i} w_j y_j} + \sum_{y \in \overline{p}} f(y)(-1)^{\sum_{j \neq i} w_j y_j} \qquad (2\ 13)$$

Comparing equations (2 10) and (2 13) we can see that

$$S'(w) = S(w) \quad \text{if} \quad w_i = 0 \quad \text{and}$$

$$S'(w) = -S(w) \quad \text{if} \quad w_i = 1$$

Hence by complementing the input variable x_i the new spectrum can be found from the original spectrum by changing the sign of the spectral coefficients having subscript i. this requires the negation of 2^{n-1} spectral coefficients

$$s_i \rightarrow -s_i$$

$$s_{ij} \rightarrow -s_{ij}$$

$$s_{ijk} \rightarrow -s_{ijk}$$

Note that s_0, s_j, s_{ij} remain unchanged

3 *Complementation of the function output*

Let S(w) be the spectrum of f(x)

and S'(w) be the spectrum of $\overline{f(x)}$

$$S'(w) = \sum_{x=0}^{x=2^n-1} \overline{f(x)}(-1)^{<ux>} \tag{2 14}$$

In binary 1, to real -1 mapping and binary 0, to real 1 mapping we have $\overline{f(x)} = -f(x)$, substituting this in the equation (2 14) we have,

$$S'(w) = \sum_{x=0}^{x=2^n-1} -f(x)(-1)^{<wx>} \tag{2 15}$$

therefore $S'(w) = -S(w)$ By negating the function output, the new spectrum is obtained by changing the sign of all coefficients This requires negation of 2^n spectral coefficients

$$s_0 \rightarrow -s_0$$

$$s_1 \rightarrow -s_1$$

$$s_2 \rightarrow -s_2$$

$$s_{12\ n} \rightarrow -s_{12\ n}$$

4 *Translation property*

Let S(w) be the spectrum of $f(x_{n-1}, x_{n-2} \quad x_i \quad x_0)$

and S'(w) be the spectrum of $f(x_{n-1} \; x_{n-2}, \quad x_i \oplus x_j \quad x_0)$

This property can be explained from the equation (2 5). For the above given matrix $\underline{g}$, x_0 is replaced by $x_0 \oplus x_1$ and x_3 is replaced by $x_3 \oplus x_2$

If x_i is replaced by $x_i \oplus x_j$, $i \neq j$ $i, j \in 1$ to n

Let $< b >$ be spectrum coefficients having subscript j the spectrum of the new function is obtained by, deleting the subscript j from $< b >$ and appending subscript j in all the coefficients This requires the interchange of 2^{n-2} pairs of coefficient values

$$s_i \leftrightarrow s_{ij}$$

$$s_{ik} \leftrightarrow s_{ijk}$$

$$s_{ikl} \leftrightarrow s_{ijkl}$$

Note that coefficients $s_0, s_j, s_{jk},$ are unchanged

5 *Disjoint translation property*

Let S(w) be the spectrum of f(x)

and S'(w) be the spectrum of $f(x) \oplus x_i$ From equation(2 8)

$$S(w) = \sum_{x \in p} f(x)(-1)^{\sum_{j \neq i} w_j x_j} + (-1)^{w_i} \sum_{x \in \bar{p}} f(x)(-1)^{\sum_{j \neq i} w_j x_j}$$

Now from the definition of WHT

$$S'(w) = \sum_{x=0}^{2^n-1} (f(x) \oplus x_i)(-1)^{<wx>} \tag{2 16}$$

writing the same way as for S(w) we have

$$S'(w) = \sum_{x=0}^{2^n-1} (f(x) \oplus x_i)(-1)^{\sum_{j \neq i} w_j x_j} + (-1)^{w_i} \sum_{x \in \bar{p}} (f(x) \oplus x_i)(-1)^{\sum_{j \neq i} w_j x_j} \tag{2 17}$$

In equation (2 17) $x_i = 0$ in the first term and $x_i = 1$ in the second In binary 1, to real -1 and binary 0, to real 1 transformation Ex-oring with binary 1 and

f(x) results negative of f(x) Making use of this property from equation (2 17) we have,

$$S'(w) = \sum_{x \in p} f(x)(-1)^{\sum_{j \neq i} w_j x_j} - (-1)^{w_i} \sum_{x \in \overline{p}} f(x)(-1)^{\sum_{j \neq i} w_j x_j} \quad (2\ 18)$$

It can also be written as,

$$S'(w) = \sum_{x \in p} f(x)(-1)^{\sum_{j \neq i} w_j x_j} + (-1)^{\overline{w}_i} \sum_{x \in \overline{p}} f(x)(-1)^{\sum_{j \neq i} w_j x_j} \quad (2\ 19)$$

comparing equations (2 8) and (2 19) we have

$$S'(w_{n-1}\ w_{n-2}, \quad , w_0) = S(w_{n-1}\ w_{n-2} \quad , \overline{w}_i \quad w_0)$$

Replacing f(x) by $f(x) \ominus x_i$, $i \in 1$ to n results in the spectrum if in every subscript of the spectral coefficients, 1 is deleted if it is present and is appended if it is not This results in the inter change of 2^{n-1} spectral coefficients

$$s_i \leftrightarrow s_0$$

$$s_{ij} \leftrightarrow s_j$$

$$s_{ijk} \leftrightarrow s_{jk}$$

Note that all 2^n coefficients are involved in this operation

2.3.2 Canonic classification

In canonic classification we arrange zeroth order and first order coefficients in descending order, they are then made positive, with s_0 being the highest magnitude coefficient [14] When we compare the spectra of many Boolean functions, spectral coefficients with identical magnitudes occur in many spectra, with changes in sign and position We map any Boolean function to its prototype Boolean function by reordering 2^n spectral coefficients by following steps

1 When maximum value coefficient lies in the primary or secondary coefficient set, generate the maximum value for s_0 using Disjoint translation property

$n \leq 2$	Spectral coefficients			
	s_0	s_1	s_2	s_{12}
1	4	0	0	0
2	2	2	2	-2

Table 2 1 Classification of Boolean functions for n $\leq$ 2

2 Generate the maximum value for all primary coefficients s_1 to s_n by making use of the translation property as necessary

3 Permute the maximum valued primary coefficients s_1 to s_n, to order them in descending order of magnitude by application of operation(1)(equation (2 5))

4 Render all primary coefficients s_0 to s_n positive by the application of operation(3) followed by operation(2) as necessary

The result of these operations is to reorder the spectral coefficient values into a positive canonic order, which represents the classification entry of the given binary function f(x) From the table (2 3) it can be seen that, all 2^{2^4} functions of $n \leq 4$ (number of variables) can be represented by eight canonic functions, and for $n \leq 5$, the 2^{5^5} possible functions, will have 48 classification entries

With all zero and first order coefficients made positive, higher order coefficients may be negative For a valid spectrum the sum of spectral coefficients must always be equal to $\pm 2^n$

2.4 Reed-Muller expansion

In the previous sections we discussed partitioning of the Boolean functions using Walsh-Hadamard transform In this section we discuss the representation of any Boolean function in Reed-Muller form, with appropriate basis [9]

It is known that any Boolean function can be realized by using AND and OR gates

$n \leq 3$	Spectral coefficients							
	S_0	S_1	S_2	S_{12}	S_3	S_{13}	S_{23}	S_{123}
1	8	0	0	0	0	0	0	0
2	6	2	2	-2	2	-2	-2	2
3	4	4	4	-4	0	0	0	0

Table 2 2 Classification of Boolean functions for $n \leq 3$

$n \leq 4$	Spectral coefficients															
	S_0	S_1	S_2	S_{12}	S_3	S_{13}	S_{23}	S_{123}	S_4	S_{14}	S_{24}	S_{124}	S_{34}	S_{134}	S_{234}	S_{1234}
1	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	14	2	2	-2	2	-2	-2	2	2	2	2	2	2	2	2	2
3	12	4	4	-4	4	-4	-4	4	0	0	0	0	0	0	0	0
4	10	6	6	-6	2	-2	-2	2	2	2	2	2	2	-2	-2	2
5	8	8	8	8	0	0	0	0	0	0	0	0	0	0	0	0
6	8	8	4	-4	4	-4	0	0	4	-4	0	0	0	0	-4	4
7	6	6	6	-2	6	-2	2	2	6	2	2	-2	2	2	2	6
8	4	4	4	4	4	4	-4	-4	4	-4	4	4	4	-4	-4	4

Table 2 3 Classification of Boolean functions for $n \leq 4$

If any function f can be represented in the form of

$$f = P_1 + P_2 + P_3 + \quad - P_k \tag{2.20}$$

or,

$$f = Q_1 \ Q_2 \ Q_3 \quad Q_m \tag{2.21}$$

where $P_i(1 \leq i \leq k)$ and $Q_j(1 \leq j \leq m)$ are arbitary functions such that $P_i \ P_j = 0$ and $\overline{Q}_i \ \overline{Q}_j = 0$ for all i and j $i \neq j$ then f can be written in the form

$$f = P_1 \oplus P_2 \oplus P_3 \quad \oplus P_k \tag{2.22}$$

or

$$f = Q_1 \odot Q_2 \odot Q_3 \quad \odot Q_m \tag{2.23}$$

Thus, any function f can be realised using EX-OR and AND gates or EX-NOR and OR gates

Any Boolean function $f(x)$ can be expressed in Reed-Muller form in three ways For example a three variable Boolean function $f(x_1, x_2 \ x_3)$ can be represented as.

1

$$f(x_1, x_2, x_3) = a_0 \oplus a_1 \ x_1' \oplus a_2 \ x_2' \oplus a_3 \ x_1' \ x_2' \oplus a_4 \ x_3' \oplus a_5 \ x_1' \ x_3' \oplus a_6 \ x_2' \ x_3' \oplus a_7 \ x_1' \ x_2' \ x_3' \tag{2.24}$$

This expansion is called complement free ring sum expansion, when all x_i' are equal to x_i(see Definition of x_i')

2 This can also be can be expanded as

$$f(x_1, x_2, x_3) = (b_0 + 0) \oplus (b_1 + x_1') \oplus (b_2 + x_2') \oplus (b_3 + x_1' + x_2') \oplus (b_4 + x_3') \oplus$$

$$(b_5 + x_1' + x_3') \oplus (b_6 + x_2' + x_3') \oplus (b_7 + x_1' + x_2' + x_3') \tag{2.25}$$

3 Similarly, a two variable function

$$f(x_1, x_2, x_3) = c_0 \oplus c_1 \ x_1 \oplus c_2 \ \overline{x}_1 \oplus c_3 \ x_2 \oplus c_4 \ x_1 \ x_2 \oplus$$

$$c_5 \ \overline{x}_1 \ x_2 \oplus c_6 \ \overline{x}_2 \oplus c_7 \ x_1 \cdot \overline{x}_2 \oplus c_8 \ \overline{x}_1 \ \overline{x}_2 \oplus$$

$$c_9\ x_3 \hat{+} c_{10}\ x_1\ x_3 \hat{+} c_{11}\ \overline{x}_1\ x_3 \hat{+} c_{12}\ x_2\ x_3 \hat{+} c_{13}\ x_1\ x_2\ x_3 \hat{+}$$

$$c_{14}\ \overline{x}_1\ x_2\ x_3 \hat{+} c_{15}\ \overline{x}_2\ x_3 \hat{+} c_{16}\ x_1\ \overline{x}_2\ x_3 \hat{+} c_{17}\ \overline{x}_1\ \overline{x}_2\ x_3 \hat{+}$$

$$c_{18}\ \overline{x}_3 \hat{+} c_{19}\ x_1\ \overline{x}_3 \hat{+}$$

$$c_{20}\ \overline{x}_1\ \overline{x}_3 \hat{+} c_{21}\ x_2\ \overline{x}_3 \hat{+} c_{22}\ x_1\ x_2\ \overline{x}_3 \hat{+}$$

$$c_{23}\ \overline{x}_1\ x_2\ \overline{x}_3 \hat{+} c_{24}\ \overline{x}_2\ \overline{x}_3 \hat{+} c_{25}\ x_1\ \overline{x}_2\ \overline{x}_3 \hat{+} c_{26}\ \overline{x}_1\ \overline{x}_2\ \overline{x}_3 \qquad (2.26)$$

Any Boolean function can also be represented in Logical equivalent canonic form in three forms, each form being dual of the corresponding RM representation

1

$$f(x_1, x_2, x_3) = (a_0 + 0) \hat{+} (a_1 + x_1) \hat{+} (a_2 + x_2) \hat{+} (a_3 + x_1 + x_2) \hat{+} (a_4 + x_3) \hat{+}$$

$$(a_5 + x_1 + x_3) \hat{+} (a_6 + x_2 + x_3) \hat{+} (a_7 + x_1 + x_2 + x_3) \qquad (2.27)$$

2

$$f(x_1, x_2, x_3) = b_0 \hat{+} b_1\ x_1 \hat{+} b_2\ x_2 \hat{+} b_3\ x_1\ x_2 \hat{+} b_4\ x_3 \hat{+} b_5\ x_1\ x_3 \hat{+} b_6\ x_2\ x_3 \hat{+} b_7\ x_1\ x_2\ x_3 \qquad (2.28)$$

3

$$f(x_1, x_2, x_3) = (c_0 + 1) \hat{+} (c_1 + x_1) \hat{+} (c_2 + \overline{x}_1) \hat{+} (c_3 + x_2) \hat{+} (c_4 + x_1 + x_2) \hat{+}$$

$$(c_5 + \overline{x}_1 + x_2) \hat{+} (c_6 + \overline{x}_2) \hat{+} (c_7 + x_1 + \overline{x}_2) \hat{+} (c_8 + \overline{x}_1 + \overline{x}_2) \hat{+}$$

$$(c_9 + x_3) \hat{+} (c_{10} + x_1 + x_3) \hat{+} (c_{11} + \overline{x}_1 + x_3) \hat{+} (c_{12} + x_2 + x_3) \hat{+} (c_{13} + x_1 + x_2 + x_3) \hat{+}$$

$$(c_{14} + \overline{x}_1 + x_2 + x_3) \hat{+} (c_{15} + \overline{x}_2 + x_3) \hat{+} (c_{16} + x_1 + \overline{x}_2 + x_3) \hat{+}$$

$$(c_{17} + \overline{x}_1 + \overline{x}_2 + x_3) \hat{+} (c_{18} + \overline{x}_3) \hat{+} (c_{19} + x_1 + \overline{x}_3) \hat{+} (c_{20} + \overline{x}_1 + \overline{x}_3) \hat{+}$$

$$(c_{21} + x_2 + \overline{x}_3) \hat{+} (c_{22} + x_1 + x_2 + \overline{x}_3) \hat{+} (c_{23} + \overline{x}_1 + x_2 + \overline{x}_3) \hat{+}$$

$$(c_{24} + \overline{x}_2 + \overline{x}_3) \hat{+} (c_{25} + x_1 + \overline{x}_2 + \overline{x}_3) \hat{+} (c_{26} + \overline{x}_1 + \overline{x}_2 + \overline{x}_3) \qquad (2.29)$$

where a_i, b_i, c_i are 1 or 0 and are different for the two representations

Any Boolean function f can be represented by a binary vector

$$f = (a_0 \; a_1 \quad a_{2^n-1})$$

$a_0, a_1 \; a_2 \quad , a_{2^n-1}$ being the coefficients of each term in equation (2.24). The function f can be obtained from binary vector and basis functions in the free ring sum expansion by using the relation

$$\underline{F} = \underline{\underline{M}} \; \underline{A} \tag{2.30}$$

where $\underline{F}$ is a coloumn vector having function output values, $\underline{A}$ is a binary coloumn vector with elements $a_0, a_1, a_2, \quad , a_{2^n-1}$ and $\underline{\underline{M}}$ is a $2^n \times 2^n$ matrix which is shown below for n = 3 each coloumn representing the basis functions, 1, $x_1, x_2 \; x_1 \; x_2 \; x_3 \; x_1 \; x_3, x_2 \; x_3, x_1 \; x_2 \; x_3$, in the same order

$$\underline{\underline{M}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The inverse of the matrix $\underline{\underline{M}}$ is the matrix itself Pre-multiplying equation (2.30) with $\underline{\underline{M}}^{-1}$ we thus have, $\underline{A} = \underline{\underline{M}} \; \underline{F}$ The matrix $\underline{\underline{M}}$ can be decomposed and is equal to $\underline{\underline{M}} = \underline{\underline{S}}^n$

For n = 3,

$$\underline{\underline{S}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

For an n variable Boolean function $f(x)$ $X' = (x_1' \; x_2' \quad x_n')$ is called polarity vector of $f(x)$ It can be seen that the polarity vector X' can take 2^n values Our problem is to determine the appropriate polarities of X' such that the binary vector $(a_0 \; a_1, a_2, \quad a_{2^n-1})$ of $f(x)$ contains minimum number of elements having values either only ones or only zeros For a function which is of the form equation (2 20) we find polarity functions for each P_i, where polarity functions are $a_0(f)$ $a_1(f)$, $a_{2^n-1}(f)$ and we substitute these back in the same equation We substitute different polarity vectors in the resultant equation to get minimum number of EX-OR gates

Chapter 3

Implementation details

In this chapter we discuss the implementation details, using the methods described in the previous chapter We considered the images having 256 intensity levels, however it can be extended to the images having any number of intensity levels Consider an image, where each pixel is represented by k-bits By selecting a single bit from the same position in the binary representation of each pixel, k-binary images called bit planes can be generated [6] For an image having 256 intensity levels we thus have 8 bit planes We have implemented two types of compression schemes *Lossless compression* which results in retrieval of exact copy of the original image This scheme has also been tested for text files *Lossy compression* which results in retrieval of approximate copy of the original image

3.1 Lossless compression

In this section we discuss the Lossless compression scheme The flow chart that has been followed is shown in figure 3 1 Our compression scheme basically consists of the following steps

- Pre processing
- Function generation
- Coding

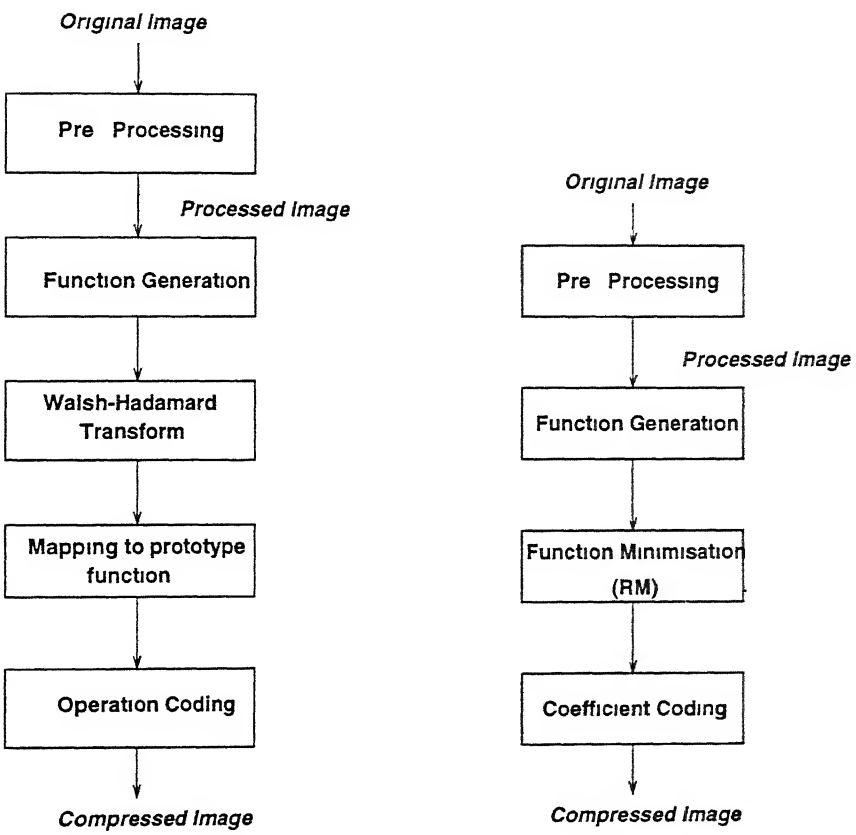


Figure 3 1 Compression scheme

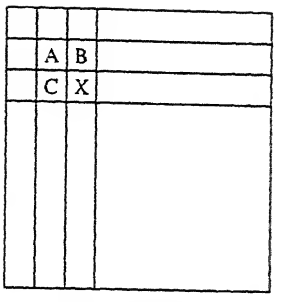


Figure 3 2 Linear prediction

3.1.1 Pre Processing

The original image file is preprocessed to get more compression Two methods for preprocessing the images have been adopted in our problem

1 Indexing method

It is possible that not all the 256 intensity levels are present in a given image The actual intensity levels appearing in the original image are arranged continuously by assigning successive integer values from '0' onwards in the pre-processed image The presence of a particular intensity level is indicated by a 256 bit overhead in the global header, to facilitate decompression

2 Linear prediction

In the Linear prediction scheme, the value of the current pixel is predicted by averaging the two pixels, one which is above the current pixel and the other to the left An error file is generated by subtracting the predicted value from the actual value as shown in figure 3 2 The first row and the first coloumn of the error file are same as the first row and first coloumn of the original image This error file is recoded by adding the absolute value of the maximum negative error, if present, to the each error value so that only positive values are present in the error file The global header also contains an extra 8 bits, to indicate the magnitude of this maximum negative error

After preprocessing the image file, these intensity levels in the pre-processed image are replaced by their equivalent gray codes So that adjacent values are mapped to adjacent codes having fewer transitions between 0 s and 1 s in the bit planes Gray coding also increases the probability of all 1(black) and all 0(white) blocks in the bit planes [6]

3.1.2 Function generation

The preprocessed image file is now split up into bit planes These bit planes can be 8 or 9 in number in case of linear predictive preprocessing If the magnitude of the error values in the error file is greater than 256, then 9 bit planes are used In the case of indexing method, the number of bit planes depends on the number of different intensity values present in the original image The global header also contains four bits to indicate the number of bit planes The bit planes are divided into fixed windows of $n \times m$ sizes, where n indicates the number of rows in the window and m indicates the number of coloumns Each window is treated as a Boolean function of $\log_2(n \times m)$ variables treating the values of $n \times m$ binary pixels as its output To indicate the number of variables involved in the Boolean function an extra four bits are used in the global header Figure 3 3 shows the conversion of a window with 4×8 binary pixels into a 5 variable Boolean function, where black represents a 1 and white represents a 0

3.1.3 Coding of the bit planes

Each bit plane is divided into windows of sizes $n \times m$ pixels The data in each window is classified into three types, all black pixels, all white pixels and mixed type(combination of both black and white) pixels Windows with all white pixels are represented as 00 and all black pixels as 11

For a mixed type of window, we try to compress the window by using Reed-Muller method of minimising and coding, and in the other technique by mapping this window into its proto type Boolean function by Walsh-Hadamard transform, and coding the operations required for it For a certain windows compression may not be achieved So the windows which are compressed are represented by a header 01 and the coded bits, which are less than number of original bits for that window The windows for which

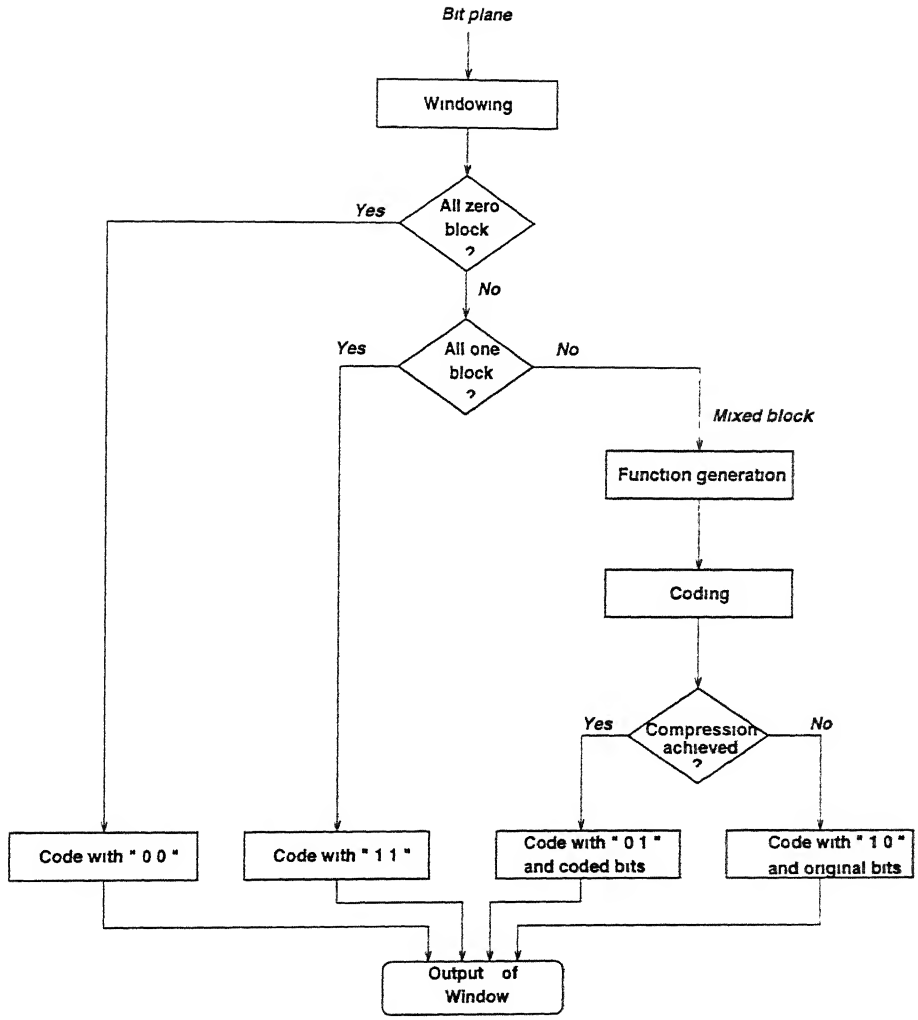


Figure 3 4 Compression of a window

compression is not achieved are represented by a header 10 and the original $n \cdot m$ bits

Coding of the mixed type window

Coding for the mixed type window can be done in two ways *Walsh-Hadamard transform* method and *Reed-Muller expansion*

1 Walsh Hadamard transform

Each mixed type window is converted into its prototype Boolean function by using $\log_2(n \times m) + 2$ sets of operations for a $n \times m$ window as described in the previous chapter The position of the prototype Boolean function in the corresponding classification table is represented using 3 or 6 bits, depending on the window size The first $log_2(n \times m) + 1$ set of operations can be represented from 0 to $n \times m$, requiring $log_2(n \times m)$ bits The last operation is represented from 0 to $2nm$ and requires $log_2(n \times m) + 1$ bits These operations are coded by adaptive block coding [10], treating each operation set as a block If the number of coded bits including the coded number of the prototype Boolean function are more than $n \times m$ bits then it is coded by a header 10 and proceeded by the original bits

For the window which is shown in the function generation diagram(figure 3 3) The window is first converted into a Boolean function $f(x_{n-1}, x_{n-2}, \ldots, x_0)$, it's Walsh-Hadamard spectrum s(w) is found, and is shown below

s_0	s_1	s_2	s_{12}	s_3	s_{13}	s_{23}	s_{123}
22	10	−10	10	−2	2	−2	2
s_4	s_{14}	s_{24}	s_{124}	s_{34}	s_{134}	s_{234}	s_{1234}
−6	6	−6	6	2	−2	2	−2
s_5	s_{15}	s_{25}	s_{125}	s_{35}	s_{135}	s_{235}	s_{1235}
−2	2	−2	2	−2	2	−2	2
s_{45}	s_{145}	s_{245}	s_{1245}	s_{345}	s_{1345}	s_{2345}	s_{12345}
2	−2	2	−2	2	−2	2	−2

The spectrum is then converted into a positive canonic Boolean function as described in the previous chapter, using the following steps

Replace input variable x_1 by $\overline{x}_1$ to make s_2 positive The new spectrum resulting is shown below

s_0	s_1	s_2	s_{12}	s_3	s_{13}	s_{23}	s_{123}
22	10	10	−10	−2	2	2	−2
s_4	s_{14}	s_{24}	s_{124}	s_{34}	s_{134}	s_{234}	s_{1234}
−6	6	6	−6	2	−2	−2	2
s_5	s_{15}	s_{25}	s_{125}	s_{35}	s_{135}	s_{235}	s_{1235}
−2	2	2	−2	−2	2	2	−2
s_{45}	s_{145}	s_{245}	s_{1245}	s_{345}	s_{1345}	s_{2345}	s_{12345}
2	−2	−2	2	2	−2	−2	2

To interchange s_4 and s_3, x_2 and x_3 are interchanged The resulting spectrum after this operation is shown below

s_0	s_1	s_2	s_{12}	s_3	s_{13}	s_{23}	s_{123}
22	10	10	−10	−6	6	6	−6
s_4	s_{14}	s_{24}	s_{124}	s_{34}	s_{134}	s_{234}	s_{1234}
−2	2	2	−2	2	−2	−2	2
s_5	s_{15}	s_{25}	s_{125}	s_{35}	s_{135}	s_{235}	s_{1235}
−2	2	2	−2	2	−2	−2	2
s_{45}	s_{145}	s_{245}	s_{1245}	s_{345}	s_{1345}	s_{2345}	s_{12345}
−2	2	2	−2	2	−2	−2	2

To make s_3, s_4 and s_5 positive, we replace input variables x_2, x_3 and x_4 by $\overline{x}_2$, $\overline{x}_3$ and $\overline{x}_4$ respectively The resultant spectrum is a positive canonic function

s_0	s_1	s_2	s_{12}	s_3	s_{13}	s_{23}	s_{123}
22	10	10	−10	6	−6	−6	6
s_4	s_{14}	s_{24}	s_{124}	s_{34}	s_{134}	s_{234}	s_{1234}
2	−2	−2	2	2	−2	−2	2
s_5	s_{15}	s_{25}	s_{125}	s_{35}	s_{135}	s_{235}	s_{1235}
2	−2	−2	2	2	−2	−2	2
s_{45}	s_{145}	s_{245}	s_{1245}	s_{345}	s_{1345}	s_{2345}	s_{12345}
−2	2	2	−2	−2	2	2	−2

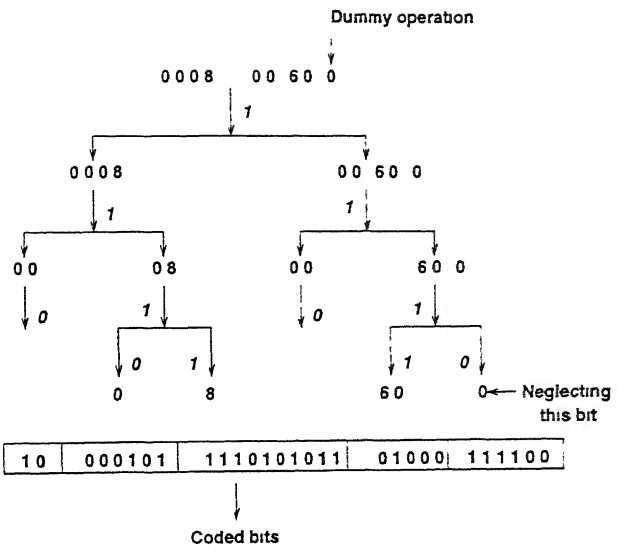


Figure 3 5 Example using WHT

To code the operation sets required, to make the window a prototype Boolean function We put a 0 if there is no operation required, in the respective operation set position Therefore for above example we get the operation set block as

0 0 0 8 0 0 60

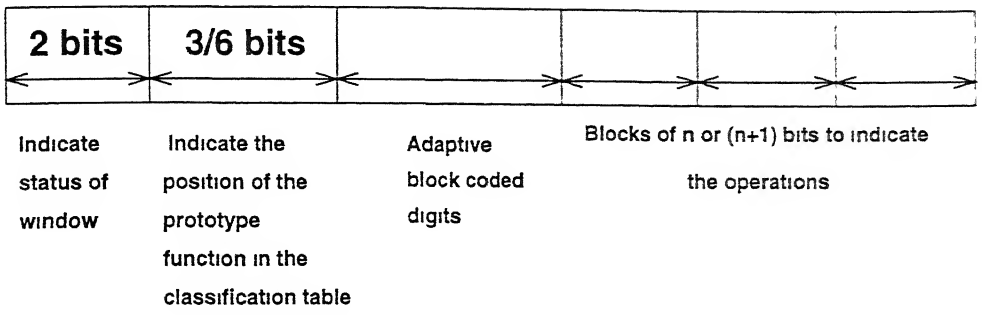
where the first block is for s_0 and so on up to last one, and the last block having number 60, is the decimal representation of 111100

x_4	x_3	x_2	x_1	x_0	$f(x)$
1	1	1	1	0	0

which indicates whether respective variables are complemented or not. 1 indicating complementation and 0 indicate uncomplement The coding of these operations is shown in figure 3 5. By treating one more dummy operation with zero value and not taking the code corresponding to that operation, the output bits of the window are 01 000101 1110101011 01000 111100 where first two bits 01 indicate that it is a compressed window and six bits 000101 indicate the position of the prototype Boolean function in the classification table for n $\leq$ 5.

2 Reed-Muller expansion

Walsh Hadamard transform :



Example for n = 5

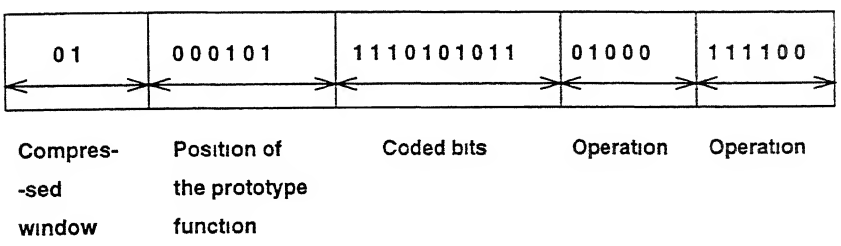


Figure 3 6 Window format for WHT

By choosing appropriate basis, each window is represented in the Reed-Muller form In this representation we try to minimise the number of coefficients having values either only ones or only zeros as explained in the previous chapter Figure 3 7 shows the general format of window in encoded form The first two bits are used to represent the window type i e , all black or all white or compressed or uncompressed window The next bit is used to indicate whether, the coded coefficients are ones or zeros Depending on the window size the upper limit c_{max}, for the number of coefficients is calculated Beyond this limit this technique leads data expansion, requiring $log_2(\frac{(n \times m)}{k}) - 1$ bits to represent the number of coefficients where $k = log_2(n \times m)$ Coding of the position of these coefficients is done by using $p \times (\log_2 (n \times m))$ number of bits, a binary 1 in the ith position indicates the presence of the variable x_i', where x_i' may be x_i or $\overline{x}_i$ and binary 0 indicates the absence of the variable

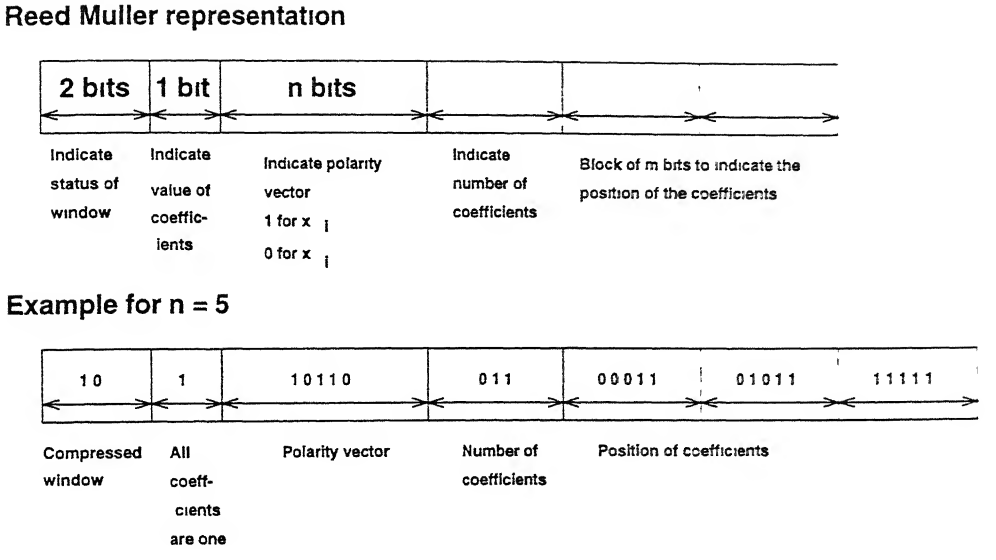


Figure 3 7 Window format and example using RMT

For the same example given in figure 3 3 If this Boolean function is expanded in uncomplement free ring form, we get

$$f(x_4, x_3, x_2, x_1, x_0) = x_0 \oplus x_0\ x_1 \ominus x_0\ x_3 \ominus x_0\ x_1\ x_2\ x_3 \ominus x_0\ x_3\ x_4 \ominus x_0\ x_1\ x_3\ x_4 \ominus x_0$$

$$x_2\ x_3\ x_4 \ominus x_0\ x_1\ x_2\ x_3\ x_4$$

having 8 coefficients with value 1 If we use polarity vector $x' = (\overline{x}_4\ x_3, \overline{x}_2\ \overline{x}_1\ x_0)$ then

$$f(x) = x_0\ \overline{x}_1 \ominus x_0\overline{x}_1\ x_3 \ominus x_0\ \overline{x}_1\ \overline{x}_2\ x_3\overline{x}_4$$

giving only three coefficients having value 1 The coded bits of this window are 01 1 10110 011 00011 01011 11111 where the first two bits 01 indicate that it is a compressed window, the next bit 1 indicates that coefficients are 1's , 10110 indicates whether the correesponding literals x_i are complemented or not, with 1 indicating complemented and 0 indicating uncomplemented form, the three bits after these bits are used to indicate the number of coefficients having value 1, and the remaining blocks of 5 bits are used to indicate the position of these coefficients

It is possible many windows in an image bit plane can be compressed by the previous methods, or by virtue of the fact that they are all black or all white windows Several

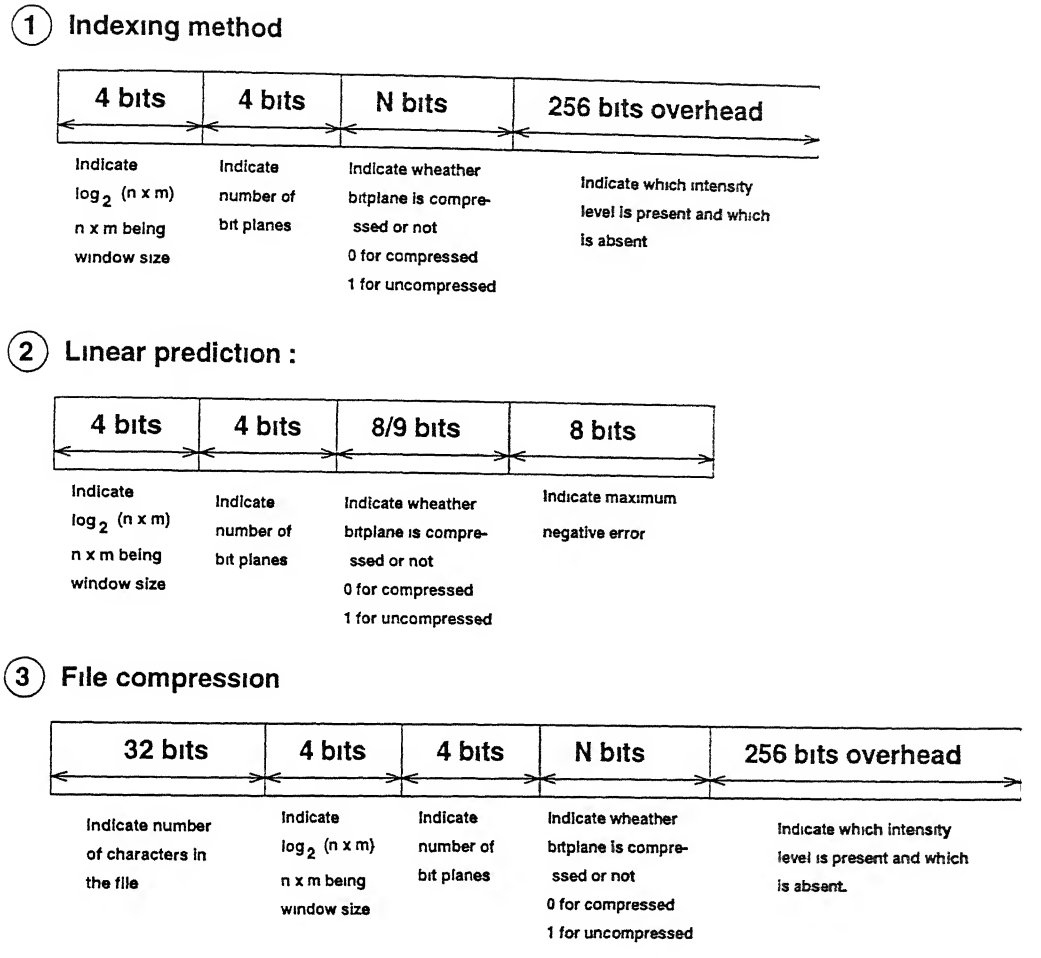


Figure 3 8 Global Header

other windows may not yield any compression, leading to a net expansion for an entire bit plane This usually happens for the bit planes obtained for two or three least significant bits of most images because they have random characteristics compared to the higher bit planes In such a case the entire bit plane is stored without carrying out any operations on it in the compressed file A one bit global header is associated with each bit plane to indicate whether the status of a bit plane

To indicate the number of bit planes four bits are used in the global header In the case of indexing method of preprocessing, global header contains a 256 bit over head to represent the presence of intensity levels in the original image We applied this lossless compression scheme to the text files and format of the global header corresponding to file compression is shown in figure 3 8

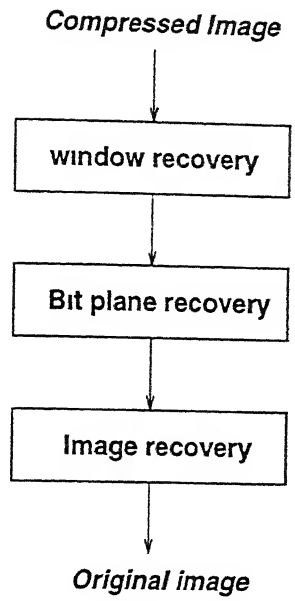


Figure 3 9 Lossless decompression Scheme

3.2 Lossless decompression

The Lossless decompression procedure consists of three steps, as shown in figure 3 9

- Window recovery
- Bit plane recovery
- Image recovery

Window recovery (decoding) consists of locating a block of data corresponding to a window in the compressed image and reconstructing the original bits of this window The two bits in the coding scheme indicate whether the window is compressed or uncompressed or all white or all black If the window is not in the compressed format then the recovery of this window is straight forward

For a compressed window, in case of Reed-Muller representation, the bit after these two bits indicate the coded coefficients are zeros or ones Depending on the value of the c_{max}, we determine the number coefficients having zero values or one values and their position From these coefficients and through the appropriate basis function, which is contained in the compressed block we reconstruct the original Boolean function By re mapping this Boolean function we get the original window

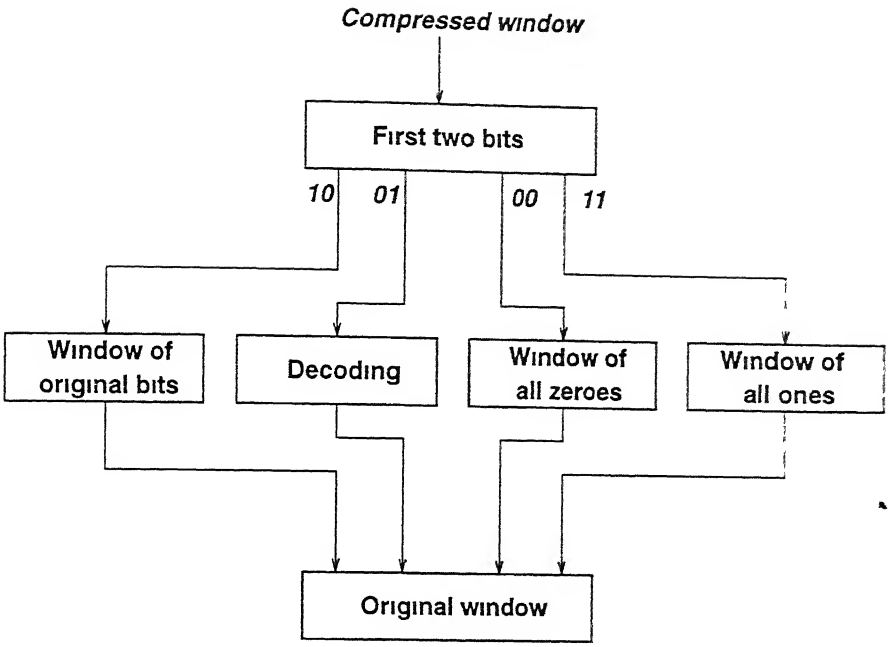


Figure 3 10 Decompression of a window

For the example which is discussed in the previous section the coded bits in case of Reed-Muller representation are 01 1 10110 011 00011 01011 11111 where 01 indicates it is compressed window, 1 indicates the coefficients are 1's 10110 is a polarity vector i' 0 indicates uncomplemented and 1 indicates complemented literal x_i, 011 indicates 3 coefficients are 1, and the positions of these coefficients are 3, 11, 31 therefore a_3, a_{11} a_{31} are 1's, using these coefficients and the polarity function, the reconstructed function

$$f(x) = x_0 \quad \overline{x}_1 \oplus x_0\overline{x}_1 \quad x_3 \oplus x_0 \quad \overline{x}_1 \quad \overline{x}_2 \quad x_3\overline{x}_4$$

from this function we find the binary pixel values, by remapping into a window

In the case of Walsh-Hadamard transform, number of the prototype Boolean function is determined and the operation set is retrieved from coded bits For the above example the coded bits are 01 000101 1110101011 01010 111100 where 01 indicates it is a compressed window and 000101 indicates the position of the prototype Boolean function in the table By decoding these coded bits we get the operation set as 0 0 0 8 0 0 60 Taking the prototype function from the table and doing operations in the reverse order we get the original spectrum Complementing the input variables $x_4, x_3, x_2,$

interchanging the input variables x_2 and x_3 and complementing x_0 we have the original spectrum By taking the inverse transform we get original Boolean function The spectrum of prototype function is

s_0	s_1	s_2	s_{12}	s_3	s_{13}	s_{23}	s_{123}
22	10	−10	10	−2	2	−2	2
s_4	s_{14}	s_{24}	s_{124}	s_{34}	s_{134}	s_{234}	s_{1234}
−6	6	−6	6	2	−2	2	−2
s_5	s_{15}	s_{25}	s_{125}	s_{35}	s_{135}	s_{235}	s_{1235}
−2	2	−2	2	−2	2	−2	2
s_{45}	s_{145}	s_{245}	s_{1245}	s_{345}	s_{1345}	s_{2345}	s_{12345}
2	−2	2	−2	2	−2	2	−2

After performing the above operation we get the following spectrum which is spectrum of original Boolean function of a window as shown below

s_0	s_1	s_2	s_{12}	s_3	s_{13}	s_{23}	s_{123}
22	10	10	−10	6	−6	−6	6
s_4	s_{14}	s_{24}	s_{124}	s_{34}	s_{134}	s_{234}	s_{1234}
2	−2	−2	2	2	−2	−2	2
s_5	s_{15}	s_{25}	s_{125}	s_{35}	s_{135}	s_{235}	s_{1235}
2	−2	−2	2	2	−2	−2	2
s_{45}	s_{145}	s_{245}	s_{1245}	s_{345}	s_{1345}	s_{2345}	s_{12345}
−2	2	2	−2	−2	2	2	−2

By taking the inverse transform of this we can reconstruct the original window

Bit plane recovery consists of reconstructing an entire bit plane from the recovered windows of this bit plane If the bit plane has not been compressed which is indicated in the global header, the recovery is straight forward

Image recovery consists of combining the individual bit planes of the image Image recovery first yields the preprocessed image file. In the case of linear prediction first error file is obtained, from which original gray values are obtained by first subtracting the maximum negative error and then using reverse prediction In the case of indexing method, from the preprocessed image file we get the original image file by substituting

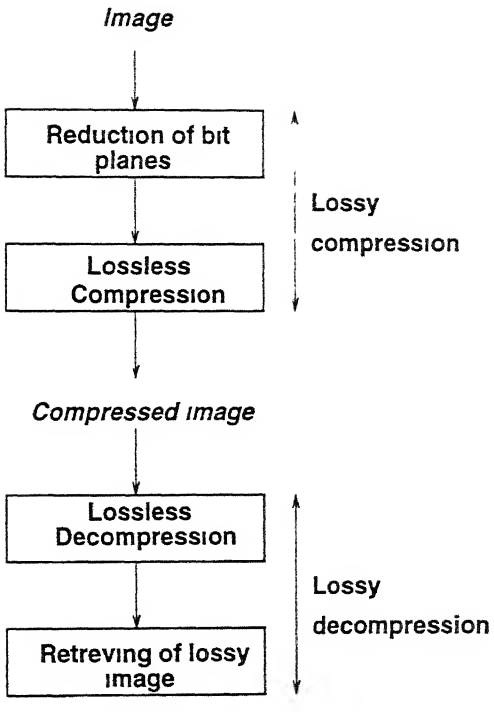


Figure 3 11 Lossy compression and decompression scheme

the original pixel values which are determined from the 256 bit overhead in the global header

3.3 Lossy compression

In this section we discuss the Lossy compression scheme, which result in retrieval of approximate copy of the original image The flow chart for this scheme is shown in figure 3 11 The lossy compression scheme consists of the following two steps

- Reducing the number of bit planes from eight to five
- Lossless compression scheme

To reduce the number of bit planes, we first reduce the number of gray levels to 32 or less in the image Here we employed the fact that human eye can not detect very small changes in gray values and have substituted a single gray value for a adjacent gray level values

We divide the range from 0 to 255 into 32 intervals 0-7, 8-15, 16-23, ..., 248-255 each interval consists of 8 gray values. Each gray value frequency is counted and for each interval we find gl_{new} by using the center of mass formula

$$gl_{new} = \frac{\sum_{i=8k}^{8k+1}(i \times f_i)}{\sum_{i=8k}^{8k+1} f_i} \quad (3.1)$$

where f_i is frequency of occurance of i'th gray level in the image and i corresponds to gray level value. These new gl_{new} gray levels are obtained for each interval and is substituted for the gray values falling into this interval. If the value obtained above is a fraction then it is rounded. Thus for each interval we find a gray value gl_{new} which is biased towards those gray levels whose frequency of occurance is more and thus help to reduce the mean square error value. The reduced image will have atmost 32 gray levels and each of these values denote the center of mass of the interval in which the gray level falls. The five MSB's of each recoded gray value represents the interval and the 3 LSB's denote the offset from start of the interval to it s the center of the mass. We can maintain these offsets in an array of 32×3 bits and recode the interval by 5 bit gray code. Thus the entire recoded image now consists of 5 bit planes and a 32×3 bit over head for denoting the offsets are included in the global header. This is the only step where we incur loss in the image. Qualitatively one can not visually distinguish between the original image and the recoded image having only 32 gray levels.

The global header in the lossy compression consists of 5 or 6 bits to denote whether the bit plane is compressed or not and a 32×3 bit over head to represent the offsets with respect to the center of mass for the 32 levels. There will be 5 or 6 bit encoded bit planes and the same window formats in the case of lossless compression.

3.4 Lossy decompression

Lossy decompression consists of the lossless decompression technique as described previously and followed by retrieving of center of mass values from the recoded gray levels and the offsets with reference to the center of mass of that level.

Chapter 4

Results

Compression and decompression algorithms have been implemented in C˙ on HP 9000/850 and tested on standard gray scale images girl, baboon, boat each of size 256×256 pixels and 8 bits per pixel(256 intensity levels) The tables give the results of the compression experiments as well as a comparision with block coding and with our schemes The results have been obtained for windows of sizes 4×4 and 4×8 for each bit plane The tables show compression ratio, compression time, decompression time for both the sizes The compression time for our compression scheme was much higher than that of block coding and decompression time is comparable to that of block coding However compression ratio is higher than that of block coding Our compression scheme and decompression scheme has been tested on some text files and the results are compared to the pack and compress commands avaliable in the HP 9000/850 The results shows that we got less compression than that of the compression commands given above The decompressed images for lossy compression were visually indistinguishable from the original images Mean square error in the decompressed lossy images is also shown in the tables Compression times reported here include the preprocessing time as well Decompression time is very less as it is very easy to generate the Boolean function from the coefficients and the basis in the case of Reed-Muller representation and for Walsh-Hadamard transform from the index of the prototype Boolean function and operations required for it, therefore it is comparable to block coding

Indexing method

IMAGE	BLOCK CODING			W.H.T.			REED MULLER		
	Comp. ratio.	Comp. time	Decomp. time	Comp. ratio.	Comp. time	Decomp. time	Comp. ratio.	Comp. time	Decomp. time
LENNA	21.25	34.90	4.80	23.64	276.30	9.24	26.38	197.0	8.50
BABOON	6.78	39.20	5.30	8.36	369.90	7.50	10.55	236.40	8.00
BOAT	15.12	38.70	5.00	17.15	329.60	8.50	20.16	211.80	8.40

Linear prediction

IMAGE	BLOCK CODING			W.H.T.			REED MULLER		
	Comp. ratio.	Comp. time	Decomp. time	Comp. ratio.	Comp. time	Decomp. time	Comp. ratio.	Comp. time	Decomp. time
LENNA	24.79	36.40	5.50	27.11	283.30	9.50	30.48	191.50	8.90
BABOON	5.85	40.90	6.50	7.83	389.70	9.50	9.53	237.10	9.40
BOAT	17.60	39.50	6.10	19.70	333.30	9.50	22.82	215.40	4.10

Table 4.1: Loss-less compression for n=5

Without pre-processing

IMAGE	M.S.E.	BLOCK CODING			W.H.T.			REED MULLER		
		Comp. ratio.	Comp. time	Decomp. time	Comp. ratio.	Comp. time	Decomp. time	Comp. ratio.	Comp. time	Decomp. time
LENNA	5.45	59.10	22.90	3.90	61.30	102.50	8.50	64.18	107.30	8.30
BABOON	5.47	43.96	23.40	5.30	45.40	188.30	7.00	47.66	140.2	7.4
BOAT	5.40	54.17	22.50	4.10	54.85	132.50	8.20	59.67	123.80	8.10

Linear prediction

IMAGE	M.S.E.	BLOCK CODING			W.H.T.			REED MULLER		
		Comp. ratio.	Comp. time	Decomp. time	Comp. ratio.	Comp. time	Decomp. time	Comp. ratio.	Comp. time	Decomp. time
LENNA	5.45	61.32	22.40	4.10	63.22	114.30	8.00	65.79	101.80	6.20
BABOON	5.47	43.06	29.30	4.40	45.00	201.80	8.90	46.64	147.30	7.10
BOAT	5.40	53.97	22.70	3.70	54.85	132.50	8.20	59.03	119.90	8.10

Table 4.2: Lossy compression for n=5

Indexing method

IMAGE	BLOCK CODING			W.H.T.			REED MULLER		
	Comp. ratio.	Comp. time	Decomp. time	Comp. ratio.	Comp. time	Decomp. time	Comp. ratio.	Comp. time	Decomp. time
LENNA	25.19	32.30	6.20	25.71	96.50	25.10	28.00	98.90	7.80
BABOON	8.76	33.40	6.00	9.13	107.50	7.80	11.27	211.80	20.16
BOAT	17.38	33.10	6.50	17.95	98.90	8.40	2035	104.80	8.30

Linear prediction

IMAGE	BLOCK CODING			W.H.T.			REED MULLER		
	Comp. ratio.	Comp. time	Decomp. time	Comp. ratio.	Comp. time	Decomp. time	Comp. ratio.	Comp. time	Decomp. time
LENNA	26.36	37.30	6.30	27.02	97.90	20.50	30.60	99.20	8.90
BABOON	6.94	40.00	7.60	7.23	122.70	9.90	9.20	124.10	10.00
BOAT	19.24	36.80	7.00	19.64	105.20	8.30	22.49	110.27	10.60

Table 4.3: Loss-less compression for n=4

Without pre-processing

IMAGE	M.S.E.	BLOCK CODING			W.H.T.			REED MULLER		
		Comp. ratio.	Comp. time	Decomp. time	Comp. ratio.	Comp. time	Decomp. time	Comp. ratio.	Comp. time	Decomp. time
LENNA	5.45	62.87	20.80	4.50	63.38	48.50	7.90	65.82	52.60	5.90
BABOON	5.47	45.80	20.20	5.00	46.25	64.7	6.20	48.40	71.90	7.20
GIRL	5.40	54.91	21.00	4.80	55.33	54.70	8.40	58.03	59.10	7.60

Linear prediction

IMAGE	M.S.E.	BLOCK CODING			W.H.T.			REED MULLER		
		Comp. ratio.	Comp. time	Decomp. time	Comp. ratio.	Comp. time	Decomp. time	Comp. ratio.	Comp. time	Decomp. time
LENNA	5.45	62.78	19.70	4.10	62.57	118.00	5.80	65.66	53.10	5.50
BABOON	5.47	44.27	26.00	5.00	45.68	73.10	10.60	46.33	76.70	7.30
GIRL	5.40	55.55	25.80	4.30	56.34	57.40	6.60	59.37	67.00	7.10

Table 4.4: Lossy compression for n=4

File name	Number of bytes	Compression ratio		
		Our scheme	Pack	Compress
One.c	12034	18.80	42.00	62.14
Two.c	7395	20.77	41.30	59.16
Three.c	4470	20.67	35.20	49.17
One.dat	306	48.36	—	61.12
Two.dat	4394	48.90	62.10	78.00
One.text	17369	20.59	45.30	57.56
Two.text	2811	21.15	41.60	46.53

Table 4.5: Different compression schemes on text files

Figure 4.1: Original image of Lenna

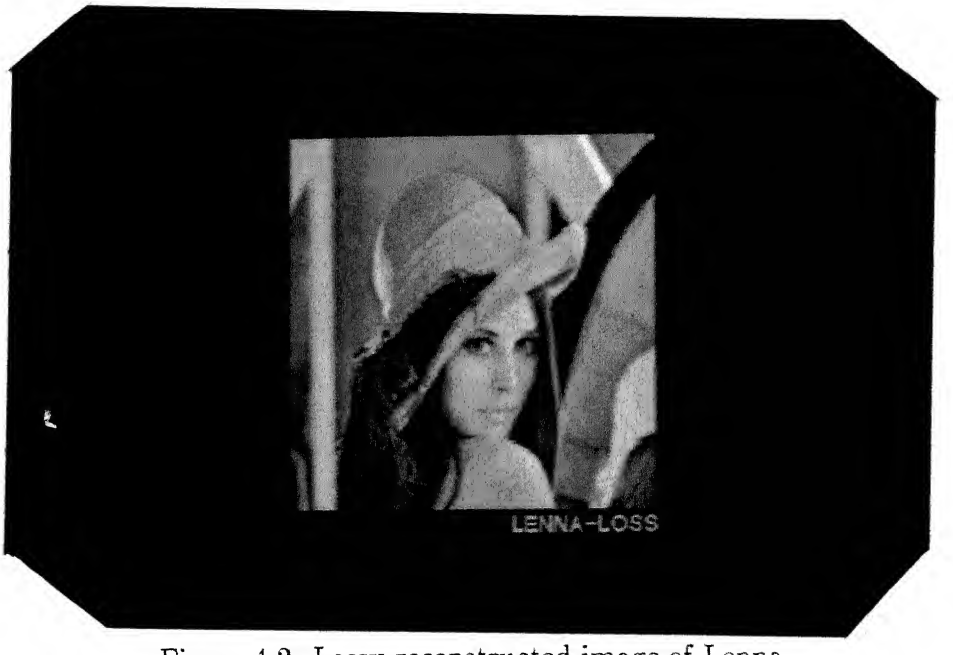


Figure 4.2: Lossy reconstructed image of Lenna

Figure 4.3: Original image of Baboon

Figure 4.4: Lossy reconstructed image of Baboon

Figure 4.5: Original image of Boat

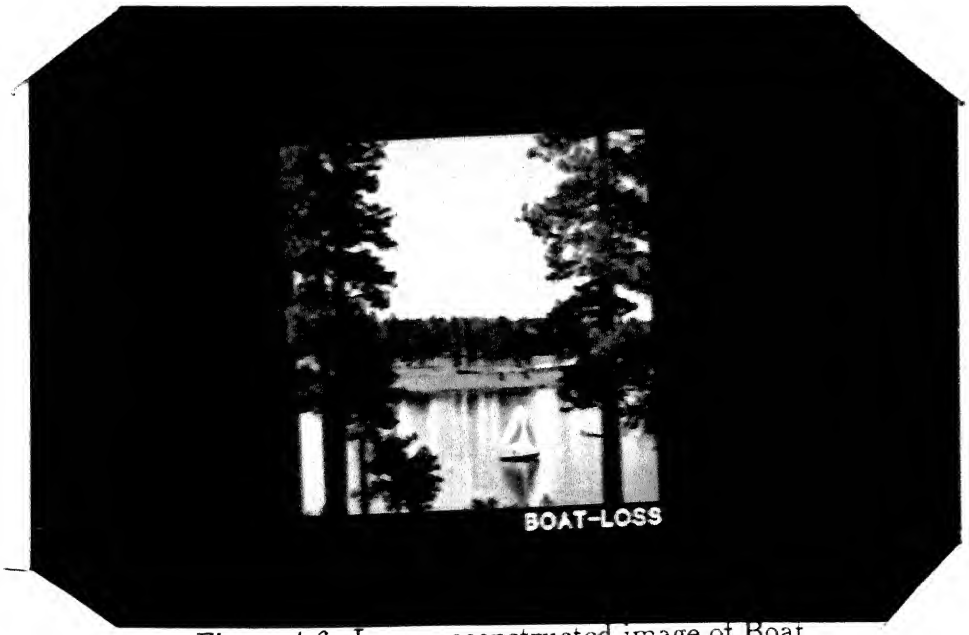


Figure 4.6: Lossy reconstructed image of Boat

Chapter 5

Conclusions and Future work

A novel lossless data compression technique for monochrome images has been presented Performance of our technique in terms of the compression ratio is better than block coding Reed-Muller way of compression is better than that of Walsh-Hadamard way of compression in terms of compression ratio and compression time Compression time is rather high, it is due to the reading of prototype function from a file for every window, and in the case of Reed-Muller representation it is due to the time taken to generate the appropriate basis(polarity vector) for each window Decompression time is comparable to that of block coding For a window of lesser sizes, compression time is lesser than that of bigger windows and the compression ratio is higher We have applied our technique to the text files but the compression ratio is lesser than that of already existing techniques It is due to the fact that other schemes employ global information to compress the file, where as our scheme employs local information to compress it We tried to apply our compression scheme to already compressed files by using pack or compress commands We got very low compression for a file having very few characters, but for files of bigger size we could not get any compression

Possible direction for the future work are, using a quad tree approach windows of different sizes can be generated this can improve the compression ratio Minimum two level Reed-Muller representation is not most compact representation for all switching function Reed -Muller like canonic forms for multi valued functions with appropriate basis functions lead to much more compact representation Higher compressions can be achieved by partitioning the multi valued functions using Walsh-Hadamard transform

and using the operations required for it The best compression scheme will be one in which the different possible way of compression methods are used and the optimal one is chosen for each block of the bit plane This can significantly improve the compression ratio at the cost of additional computation

Bibliography

[1] S L Hurst, J C Mujio, D M Muller Spectral Techniques in Digital Logic' *Academic Press, Inc , New York*, 1985

[2] S L Hurst, 'The Logical Processing of Digital Signals', *Crane Russak and company Inc* , 1978

[3] M G Karpovsky, Finite Orthogonal Series in the Design of Digital Devices (Analysis, Synthesis and Optimization)', *John Wiley and Sons, New York* 1976

[4] D C Kay, J R Levine, "Graphics file formats', *Windcrest/Mc Graw-Hill* 1992

[5] J Augustine, W feng, J Jacob, 'Logic minimization based approach for compressing image data", *Proc of the 8th international conf on VLSI design*, pp 225-228, Jan 1995

[6] M Kunt, O Johnsen , Block coding of graphics, a tutorial review', *Proc IEEE* , Vol 68, No 7, pp 770-786, July 1980

[7] P Franti, "A fast and efficient compression method for binary images' *Signal processing, image communication* , Vol 6 No 1 pp 69-76 Mar 1994

[8] T A Welch, 'A technic for high performance data compression", *IEEE Computer*, Vol 17, No 6, June 1984

[9] A Mukhopadyay, G Schmitz, "Minimization of EX-OR and logical equivalence switching circuits", *IEEE Trans computer* , Vol C-19, pp 132-140, 1970

[10] T S Haung, "Coding of two tone images", *IEEE Trans on communications* , Vol Com 25, No 11, pp 1406-1424, Nov 1977

[11] I H Witten, Arithmatic coding for data compression', *Comm of the ACM* Vol 30, pp 520-540, June 1987

[12] G G Langedon,J Rissanen, Compression of black and white images with arithmetic coding' *IEEE Trans on comm* , Vol 29, No 6 pp 858-867 July 1981

[13] E Zehavi, J K Wolf, On run length codes ' *IEEE Trans on information theory* , Vol 34, No 1 pp 45-54, Jan 1988

[14] C R Edwards, The application of Rademacher-Walsh transform to Boolean function and its application to threshold logic synthesis , *IEEE Trans computer*, Vol C-24, pp 48-62, 1975